第5章 VC目标代码的阅读理解

*5.1 汇编语言形式的目标代码

*5.2 C语言部分编译的解析

*5.3 C++部分功能实现细节

5.4 目标程序的优化

*5.5 C库函数分析

*5.6 C程序的目标代码分析

5.4 目标程序的优化

5.4.1 关于程序优化

5.4.2 使大小最小化

5.4.3 使速度最大化

5.4.4 内存地址对齐

5.4.1 关于程序优化

➢关于优化

- ✓ 优化就是提高目标程序的效率，体现在“时间”和“空间”两个方面。在时间方面是执行速度最大化，在空间方面是占用空间最小化。
- ✓ 在时间和空间两个方面的效率同时得到提高是最好。但时间和空间常常矛盾。空间换时间，或者时间换空间。
- ✓ 优化的关键是算法优化。
- ✓ 从汇编语言的角度看，主要指利用恰当的指令。

这里从汇编语言的角度介绍目标程序的优化，
假设：算法已经优化，或算法已经确定。

5.4.1 关于程序优化

➢关于优化

清寄存器EBX

✓ 有多种不同方法实现同一功能

```
MOV    EBX, 0          ;5字节
XOR    EBX, EBX        ;2字节
SUB    EBX, EBX        ;2字节
AND    EBX, 0          ;3字节
```

采用哪条指令比较好，与具体的场合有关

5.4.1 关于程序优化

➢关于优化

- ✓ 一般而言，采用相同的算法，由汇编语言编写的程序效率最高。因为汇编语言更能充分发挥机器的特性。但是，用汇编语言编程的工作效率却是最低。

代价！

- ✓ 实际上，现在高级语言的编译器功能很强劲，由编译器生成的目标代码已经“足够好”，或者说好过普通汇编语言程序员编写的程序。

某种意义上，这也是越来越少使用汇编语言编写源程序的原因之一。

5.4.1 关于程序优化

➢演示函数cf520

演示VC编译器的优化工作

```
unsigned  int  cf520(unsigned  char  n)
{
    unsigned  int  x, y, sum;
    x = n * 8;
    y = n / 8;
    sum = x + y;
    return  sum;
}
```

5.4.1 关于程序优化

➢演示函数cf520的目标代码

演示VC编译器的优化工作

速度最大化
大小最小化

```
_n$ = 8
cf520    PROC
    push   ebp
    mov    ebp, esp
    movzx  eax, BYTE PTR _n$[ebp]          ;x = n
    mov    ecx, eax                        x
    shr    ecx, 3                          n/8
    lea    eax, DWORD PTR [ecx+eax*8]      ;sum = y + x*8
    pop    ebp
    ret
cf520    ENDP
```

cf520(unsigned char n)
自动类型转换

移位指令代替除法指令

乘法、加法，合并进行

5.4.1 关于程序优化

➢演示函数cf520的目标代码（另一种）

演示VC编译器的优化工作

不建立堆栈框架

速度最大化
大小最小化

```
cf520    PROC
    movzx eax, BYTE PTR [esp+4]          ;x = n
    mov   ecx, eax                       ;y = x
    shr   ecx, 3                         ;y = n/8
    lea   eax, DWORD PTR [ecx+eax*8]     ;sum = y + x*8
    ret
cf520    ENDP
```

演示！

VC2010的编译器相当“聪明”。
不仅用寄存器作为局部变量，而且还充分利用
IA-32系列处理器的相关指令。
这样的目标代码在“时空”两个方面都是高效的。

5.4.1 关于程序优化

➢关于优化

✓ 用寄存器作为局部变量能大大提高效率

- 寄存器位于CPU内部，存取寄存器速度最快；
- 表示寄存器的编码比较短，相应指令的长度也就比较短。

✓ 优化与处理器关系密切

- 优化依赖于处理器。

5.4.2 使大小最小化

➢关于大小最小化

- ✓ “使大小最小化”，就是使得目标程序长度最短，也即把组成目标程序的所有指令长度相加最小。
- ✓ IA-32系列处理器属于复杂指令系统的处理器，其指令长度少则1字节，多则超过10字节。
- ✓ 大小最小化的方法
  - 采用寄存器作为变量
  - 采用长度较短的指令或者指令片段

5.4.2 使大小最小化

➢演示函数cf37

计算1到n之间的整数之和

```
int  cf37(int  n)
{
    int  i, sum;
    sum = 0;
    for ( i=1; i <= n; i++ )
        sum += i;
    return  sum;
}
```

5.4.2 使大小最小化

➢演示函数cf37的目标代码（大小最小化）

计算1到n之间的整数之和

```
    push   ebp
    mov    ebp, esp
    xor    ecx, ecx                ;33 C9
    inc    ecx                     ;41
    xor    eax, eax                ;33 C0
    cmp    DWORD PTR _n$[ebp], ecx
    jl     SHORT LN1@cf37
LL3@cf37:
    add    eax, ecx                ;03 C1
    inc    ecx                     ;41
    cmp    ecx, DWORD PTR _n$[ebp]
    jle    SHORT  LL3@cf37
LN1@cf37:
    pop    ebp
    ret
```

寄存器作为变量

变量sum由EAX表示

变量i由ECX表示

_n$ = 8

5.4.2 使大小最小化

➢演示函数cf37的目标代码（大小最小化）

计算1到n之间的整数之和

利用长度较短指令
或者代码片段

```
    push   ebp
    mov    ebp, esp
    xor    ecx, ecx                ;33 C9
    inc    ecx                     ;41
    xor    eax, eax                ;33 C0
    cmp    DWORD PTR _n$[ebp], ecx
    jl     SHORT LN1@cf37
LL3@cf37:
    add    eax, ecx                ;03 C1
    inc    ecx                     ;41
    cmp    ecx, DWORD PTR _n$[ebp]
    jle    SHORT  LL3@cf37
LN1@cf37:
    pop    ebp
    ret
```

mov ecx,1

mov eax,0

add ecx,1

5.4.2 使大小最小化

➢演示函数cf521

根据年份判断某年是否为闰年

```
int  cf521(unsigned  int  year)
{
    int  leap = 0;
    if (((year % 4 == 0) && (year % 100!=0)) || (year % 400==0))
    leap = 1;
    return  leap;
}
```

5.4.2 使大小最小化

➢演示函数cf521的目标代码（大小最小化）

```
_year$ = 8
cf521     PROC
      push   ebp
      mov    ebp, esp
      xor    ecx, ecx
      test   BYTE PTR _year$[ebp], 3
      push   esi
      jne    SHORT  LN1@cf521
      mov    eax, DWORD PTR _year$[ebp]  ;EAX = year
      push   100
      xor    edx, edx
      pop    esi
      div    esi
      test   edx, edx
      jne    SHORT LN2@cf521
```

寄存器作为变量
变量leap由ECX表示

利用长度较短指令
或者代码片段

year被4整除吗？

mov esi, 100

year被100整除吗？

5.4.2 使大小最小化

➢演示函数cf521的目标代码（续）

大小最小化

```
LN1@cf521:
    mov    eax, DWORD PTR _year$[ebp]  ;EAX = year
    xor    edx, edx
    mov    esi, 400
    div    esi
    test   edx, edx
    jne    SHORT LN3@cf521
LN2@cf521:
    xor    ecx, ecx
    inc    ecx
LN3@cf521:
    mov    eax, ecx
    pop    esi
    pop    ebp
    ret
```

year被400整除吗？

优化依赖处理器！

leap=1

5.4.3 使速度最大化

➢关于速度最大化

✓ “使速度最大化”就是使得执行目标程序的速度最快。

✓ 影响目标程序执行速度的因素：

- 指令执行的时钟数
- 高速缓存（cache）的命中
- 指令执行流水线及其配对
- 等等

5.4.3 使速度最大化

➢关于速度最大化

✓ 速度最大化方法：

除法指令时钟数多

- 避免时钟数多的指令
- 减少转移指令

影响执行流水线

- 减少循环执行次数
- 存储器地址对齐
- 等等

5.4.3 使速度最大化

➢演示函数cf521

根据年份判断某年是否为闰年

分析速度最大化的目标代码

```
int  cf521(unsigned  int  year)
{
    int  leap = 0;
    if (((year % 4 == 0) && (year % 100!=0)) || (year % 400==0))
    leap = 1;
    return  leap;
}
```

5.4.3 使速度最大化

➢演示函数cf521的目标代码（速度最大化）

```
_year$ = 8
cf521    PROC
    push   ebp
    mov    ebp, esp
    mov    ecx, DWORD PTR _year$[ebp]    ;ECX作为参数year
    push   esi
    xor    esi, esi   ;ESI作为变量leap
    test   cl, 3
    jne    SHORT LN1@cf521
    mov    eax, 1374389535
    mul    ecx
    shr    edx, 5
    imul   edx, 100
    mov    eax, ecx
    sub    eax, edx
    jne    SHORT LN2@cf521                ;
```

避免除法指令

(year % 4)==0 ?

EDX = year / 100

year % 100
EAX = year-(year/100)*100

5.4.3 使速度最大化

➢演示函数cf521的目标代码（速度最大化）

```
LN1@cf521:
    mov    eax, 1374389535
    mul    ecx
    shr    edx, 7
    imul   edx, 400
    sub    ecx, edx
    jne    SHORT LN6@cf521
LN2@cf521:
    mov    eax, 1
    pop    esi
    pop    ebp
    ret
LN6@cf521:
    mov    eax, esi
    pop    esi
    pop    ebp
    ret
```

避免除法指令

year % 400
ECX = year-(year/400)*400

leap = 1

减少转移指令

leap = 0

5.4.3 使速度最大化

➢演示函数cf37

计算1到n之间的整数之和

```
int  cf37(int  n)
{
    int  i, sum;
    sum = 0;
    for ( i=1; i <= n; i++ )
        sum += i;
    return  sum;
}
```

作为演示，
在3.1.3介绍过禁止优化的目标代码，
在5.4.2介绍过大小最小化的目标代码，
现在观察速度最大化的目标代码

5.4.3 使速度最大化

➢演示函数cf37的目标代码（速度最大化）

```
_n$ = 8
cf37    PROC
    push   ebp
    mov    ebp, esp
    push   ebx
    push   edi
    ;
    mov    edi, DWORD PTR _n$[ebp]   ;EDI存放n
    xor    edx, edx                  ;EDX作为sum1，清0
    xor    ecx, ecx                  ;ECX作为sum2，清0
    xor    ebx, ebx                  ;EBX作为“零头”
    lea    eax, DWORD PTR [edx+1]    ;EAX作为i，i=1
    cmp    edi, 2                    ;循环次数n太小？
    jl     SHORT LC9@cf37            ;确实太小，则转
```

寄存器作为变量

5.4.3 使速度最大化

➢演示函数cf37的目标代码（续）

减少循环

```
    push   esi
    lea    esi, DWORD PTR [edi-1]        ;ESI相当于（n-1）
    npad   6
LL10@cf37:
    add    edx, eax                      ;sum1 += i
    lea    ecx, DWORD PTR [ecx+eax+1]    ;sum2 += (i+1)
    add    eax, 2                        ;i = i+2
    cmp    eax, esi                      ;i <= n-1 ?
    jle    SHORT LL10@cf37               ;是，继续循环
    pop    esi
LC9@cf37:
    cmp    eax, edi                      ;i > n ?
    jg     SHORT LN8@cf37                ;是，跳转
    mov    ebx, eax                      ;准备“零头”
LN8@cf37:
```

伪指令！为了地址对齐

地址对齐

循环体内：重复累加操作

5.4.3 使速度最大化

➢演示函数cf37的目标代码（续二）

```
LN8@cf37:
    lea    eax, DWORD PTR [ecx+edx]      ;EAX = sum1+sum2
    pop    edi
    add    eax, ebx    ;加上可能存在的“零头”
    pop    ebx
    pop    ebp
    ret
cf37    ENDP
```

5.4.4 内存地址对齐

➢关于内存地址对齐

- ✓ **内存地址对齐**指，访问存储单元的地址是存储单元尺寸（字节数）的倍数。例如，访问某双字存储单元，那么当地址是4的倍数时，就是对齐的。
- ✓ 在采用IA-32系列处理器的系统中，存储器的读写地址必须是4的倍数。如果不是双字地址对齐，那么将自动分解为两次读写操作，导致多读写操作一次。

5.4.4 内存地址对齐

➢关于内存地址对齐

✓ 示例：

```
MOV    EAX, [0000137FH]        ;地址不对齐
MOV    EAX, [00001380H]        ;地址对齐
```

第一条指令读存储器的操作分解为：

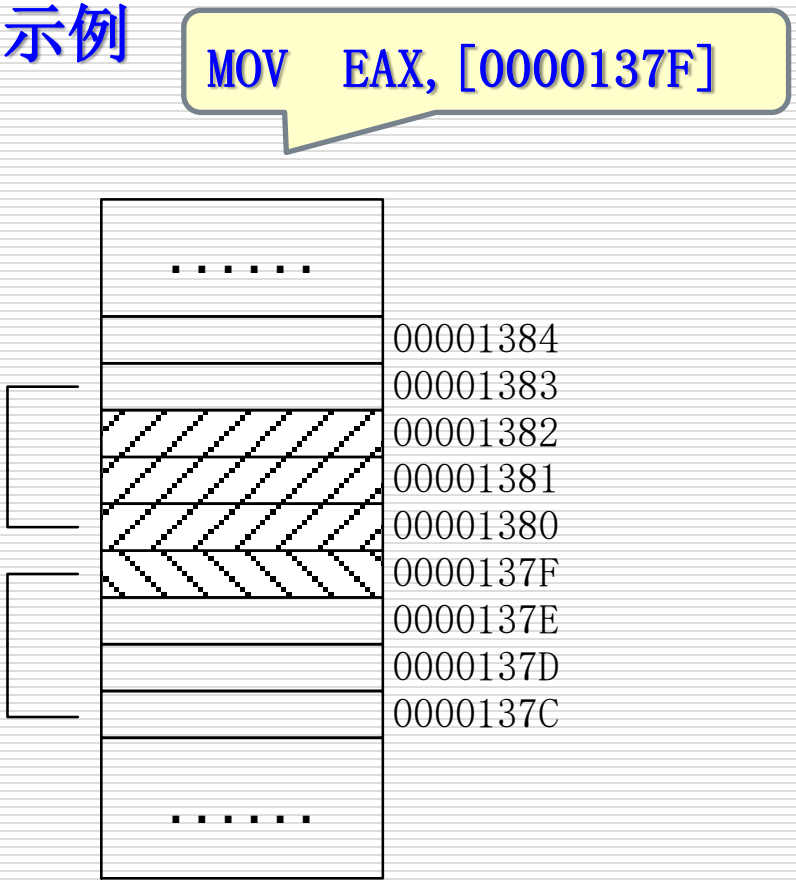
读地址为[0000137CH]的双字

读地址[00001380H]的双字

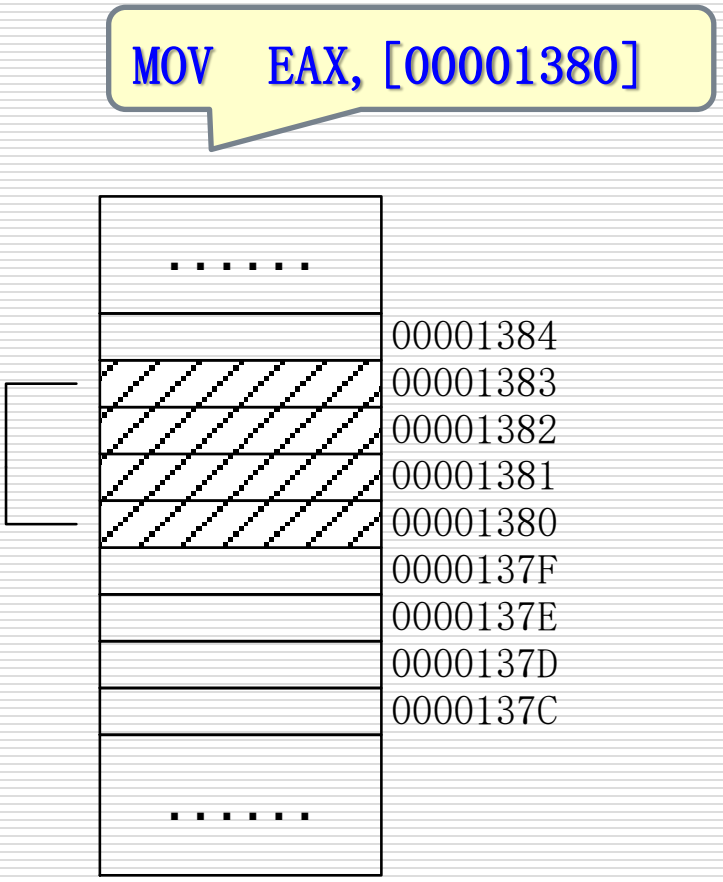
形成地址为[0000137FH]的双字

5.4.4 内存地址对齐

- 关于内存地址对齐
  - ✓ 示例



（a）读[0000137F]双字



（b）读[00001380]双字